

A multi-instance learning algorithm based on a stacked ensemble of lazy learners

Ramasubramanian Sundararajan

RAMASUBRAMANIAN.SUNDARARAJAN@GE.COM

Hima Patel

HIMA.PATEL@GE.COM

Manisha Srivastava*

MANISHA.S.SRIVASTAVA@GMAIL.COM

GE Global Research

John F. Welch Technology Centre, 122 EPIP, Whitefield Road, Bangalore 560066, India

Abstract

This document describes a novel learning algorithm that classifies “bags” of instances rather than individual instances. A bag is labeled positive if it contains at least one positive instance (which may or may not be specifically identified), and negative otherwise. This class of problems is known as multi-instance learning problems, and is useful in situations where the class label at an instance level may be unavailable or imprecise or difficult to obtain, or in situations where the problem is naturally posed as one of classifying instance groups. The algorithm described here is an ensemble-based method, wherein the members of the ensemble are lazy learning classifiers learnt using the Citation Nearest Neighbour method. Diversity among the ensemble members is achieved by optimizing their parameters using a multi-objective optimization method, with the objectives being to maximize Class 1 accuracy and minimize false positive rate. The method has been found to be effective on the *Musk1* benchmark dataset.

1. Introduction

This document describes a novel learning algorithm that classifies “bags” of instances rather than individual instances. A bag is labeled positive if it contains at least one positive instance (which may or may not be specifically identified), and negative otherwise. This class of problems is known as multi-instance learning (MIL) problems.

This setting is applicable in a number of problems where traditional two-class classifiers may face one or more of the following difficulties:

* This work was done when this author was employed at GE Global Research

Precise labeling unavailable Getting precisely labeled instances is difficult or time-consuming, whereas a precise labeling at a lower level of granularity can be more easily obtained for a larger sample of instances.

Consider the problem of automatically identifying patients who suffer from a certain ailment, based on detection of certain abnormalities in medical images acquired through any appropriate modality (X-ray, CT, MRI, microscopy etc.). Building a model for such automatic identification usually involves creating a labeled sample for training, i.e., having an expert mark out these abnormalities in patients who have said ailment, and creating a training dataset that contains both these images as well as those from normal patients. The model itself is usually some sort of classifier that operates either on images, or regions of interest thereof, or on individual pixels. This method is very well understood and applied in practice.

However, while obtaining ground truth, i.e., labeled examples, certain practical difficulties exist. For instance, the expert may not mark all abnormalities in an image comprehensively and accurately. For instance, if the expert is marking out pink/red coloured rod-shaped objects in a microscopy image of a sputum smear slide to indicate the presence of *Mycobacterium Tuberculosis*, he/she may just mark a few to convey that the patient has the disease, rather than marking every one of them. Also, the marking may be a single pixel inside the object, or an approximate bounding box, rather than a perfect contour of the bacterium. In some cases, the expert may simply mark the image/patient as abnormal rather than mark out the specific region of abnormality, especially in cases where the abnormal region is of diffuse character (e.g. late stage Pneumoconiosis on PA chest x-rays).

These practical issues consequently introduce some label noise in the data, either through unmarked or approximately marked abnormalities. The level of granularity at which the label can be considered reliable is the image/patient itself.

From a traditional classification approach, this throws up two options:

1. *Learn at a pixel/ROI level in the presence of label noise.* While some classifiers are relatively robust to label noise, their accuracy is generally poorer than when they are learnt without noise. This means that, in cases where the patient has very few regions of abnormality, any error will lead to a misdiagnosis, which is not ideal.
2. *Learn at a patient level,* by characterizing each image (or set of images corresponding to a patient) using a single feature set and then training the classifier using these features and the image-level class labels. The trouble with this approach is that the features themselves are likely to characterize both normal and abnormal regions of the image;

depending on the size of the abnormality relative to the size of the image and the nature of the features themselves, the performance of the classifier built using this approach is likely to vary considerably.

A third option, which is proposed in the CAD literature pertaining to multi-instance learning, is to consider each image (or images pertaining to a patient) as an instance bag. The individual instances in the bag may either be pixels or regions of interest (identified using a method with high sensitivity but not necessarily a low false positive rate). These instance bags are labeled using the patient label. In cases where some of the pixels or regions of interest are reliably labeled, we may be able to use this additional information as well.

Classifying sequences The problem itself is one of classifying a bag, or sequence of instances rather than one of classifying a single instance. Examples include prognostics applications, wherein one may wish to predict whether or not a sequence of events or equipment states is likely to lead to a fault. In this case, although the historical data may have precise labeling as to when the fault occurred, the problem itself is one of classifying a sequence (or bag) of machine states. Since the data on which the model is learnt or deployed may not contain time-continuous sequences of states for a particular unit (i.e., data for some periods leading to the fault may be missing), modeling the problem as one of instance bag classification may be a more natural alternative.

The algorithm described in this document is an ensemble-based method, in which the members of the ensemble are lazy learning classifiers learnt using the Citation Nearest Neighbour method. Diversity among the ensemble members is achieved by optimizing their parameters using a multi-objective optimization algorithm, with the objectives being to maximize Class 1 accuracy and minimize false positive rate.

The organization of this document is as follows: Section 2 briefly describes the prior work in this area. Section 3 describes the proposed method. Section 4 describes an application of this method to a benchmark dataset, along with some results. Section 5 concludes with some directions for further work.

2. Literature Survey

The MIL problem was first discussed in [Dietterich et al. \(1997\)](#), who also proposed a learning algorithm based on axis-parallel rectangles to solve it. Subsequently, a number of other researchers in the area have proposed MIL algorithms, notably [Wang and Zucker \(2000\)](#); [Zhang and Zhou \(2009\)](#); [Zhou and Zhang \(2007\)](#); [Maron and Lozano-Perez \(1998\)](#); [Viola et al. \(2006\)](#); [Andrews](#)

et al. (2002). For a survey of MIL algorithms, please refer to Babenko (2008); Zhou (2004). In our algorithm, we specifically focus on extending the lazy learning approach proposed in Wang and Zucker (2000).

Applications of MIL to real-life problems have been explored in the literature as well. Examples are primarily to be found in the computer-aided diagnostics and image classification area in the form of both papers (D. Wu and Boyer (2009); Fung et al. (2007); Bi and Liang (2007)) and patents (Bi and Liang (2010); Krishnapuram et al. (2009); Rao et al. (2011)).

The idea of using ensembles for classification in general has been explored extensively in the literature. While Zhang and Zhou (2009); Zhou and Zhang (2007) discuss the use of ensembles in the MIL context, the method by which the ensemble elements are combined is fairly straightforward (simple voting scheme). Bi and Liang (2007) propose the use of a cascaded ensemble of linear SVMs combined using an AND-OR framework, but with the key difference that all elements in the cascade are optimized simultaneously and the execution order of the classifiers is not decided *a priori*.

We approach the ensemble construction problem from the generalized standpoint suggested in Wolpert (1992) – it is possible that each classifier in the ensemble has learnt a different aspect of the underlying problem; however, combining them may require an additional layer of complexity. We therefore use the stacked generalization approach Wolpert (1992), wherein a second layer classifier is used to combine the outputs of the first layer ensemble of classifiers. This second layer classifier operates like a single-instance learner, and can therefore be built using any of a variety of standard classifier methods, such as support vector machines, random forests and so on Breiman et al. (1984); Breiman and Schapire (2001); Cristianini and Shawe-Taylor (2000).

3. Methodology

Consider a set of instance bags $\{B^1, B^2, \dots, B^N\}$, where each bag B^i contains instances $\{X_1^i, \dots, X_{N_i}^i\}$ and is labeled $Y^i \in \{-1, +1\}$ – we shall refer to them as positive and negative bags. The specific instance-level labels y_j^i for X_j^i may be unknown, except that Y^i is set to 1 if at least one of Y_j^i is 1, and -1 otherwise. The task of the proposed algorithm is to predict the true label Y^{new} for an unseen bag B^{new} . Let the prediction be denoted by \hat{Y}^{new}

The broad steps followed by the proposed classifier are as follows (see figure 1):

1. Use an ensemble of multi-instance classifiers that use the Citation Nearest Neighbour technique. Let the classifiers be denoted by $C_{(j)}, j = 1 \dots J$, and the predictions from these classifiers for B^{new} be denoted by $\hat{Y}_{(j)}^{new} = C_{(j)}(B^{new})$. Each classifier uses a different set of parameters, so that diversity among the ensemble is maintained.

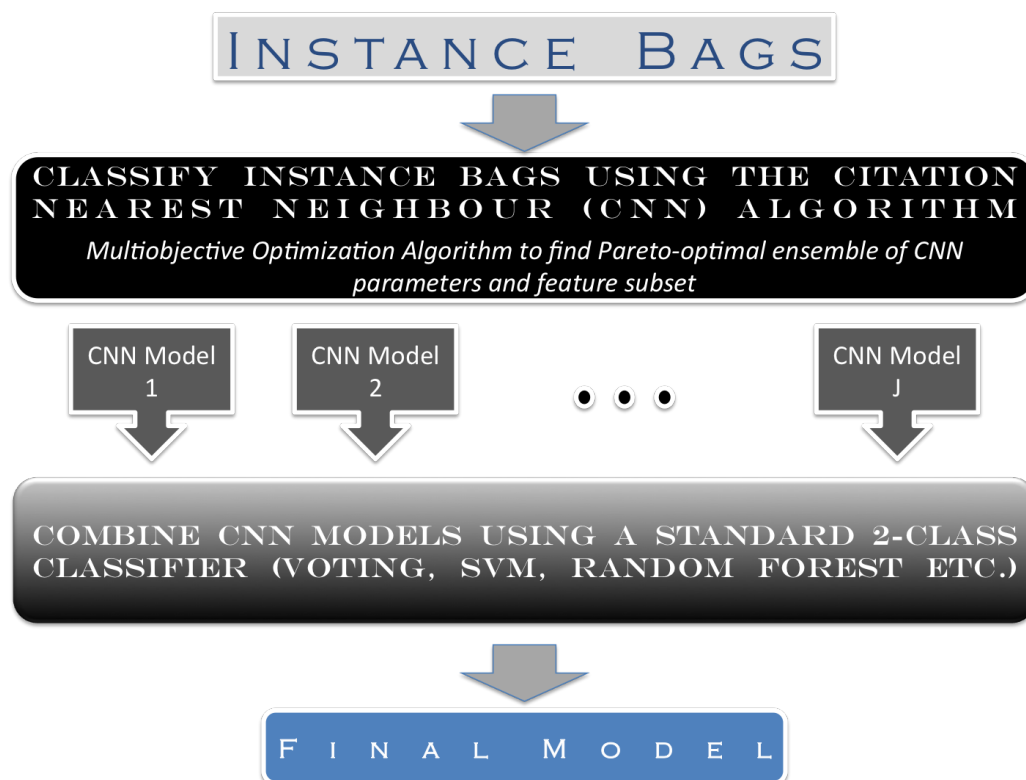


Figure 1: Proposed Multi-Instance Learning Classifier

2. Combine the predictions $\hat{Y}_{(1)}^{new}, \dots, \hat{Y}_{(J)}^{new}$ using a normal classifier F and return the final prediction, i.e., $\hat{Y}^{new} = F(\hat{Y}_{(1)}^{new}, \dots, \hat{Y}_{(J)}^{new})$.

Algorithm 1: Prediction on an unseen instance bag

Input: *Training sample:* $Train = \{B^1, B^2, \dots, B^N\}$, where each bag B^i contains instances $\{X_1^i, \dots, X_{N_i}^i\}$ and is labeled $Y^i \in \{-1, +1\}$

CNN classifiers: $C_{(1)} \dots C_{(J)}$

Final classifier: F

Unseen instance bag: B^{new}

Output: *Prediction for B^{new} :* \hat{Y}^{new}

begin

1. Apply each CNN classifier $C_{(j)}, j = 1 \dots J$ to B^{new} . Let

$$\hat{Y}_{(j)}^{new} = C_{(j)}(Train, B^{new})$$

be the prediction from the j^{th} CNN classifier.

2. Combine the predictions using the final classifier F . Let

$$\hat{Y}^{new} = F(\hat{Y}_{(1)}^{new}, \dots, \hat{Y}_{(J)}^{new})$$

be the final prediction for B^{new} .

3. **Return** \hat{Y}^{new}

end

In the following subsections, we describe how this classifier is built. Section 3.1 describes the Citation Nearest Neighbour (CNN) classifier for multi-instance learning. Section 3.2 describes how an ensemble of CNN classifiers is built, and the predictions of the ensemble combined to get the final prediction.

3.1. Citation Nearest Neighbour Algorithm

Our proposed algorithm uses a customized version of a simple, yet effective lazy learning method, namely the Citation Nearest Neighbour (CNN) technique, originally proposed in Wang and Zucker (2000).

The CNN technique is simply a nearest neighbour technique with an additional inversion step. We find *references*, i.e., the neighbours of a test bag and note their labels. Similarly, we find *citers*, i.e., those training bags that would consider this test bag a neighbour and note their labels as well. We then compare the number of positive versus negative bags in this calculation and arrive at a final result based on whether the positive bags in references and citers put together outnumber the negative bags.

The distance between bags is normally calculated using a metric called the minimal Hausdorff distance. Given two bags $A^1 = \{a_1^1 \dots a_p^1\}$ and $A^2 = \{a_1^2 \dots a_q^2\}$, the Hausdorff distance is defined as:

$$H(A^1, A^2) = \max(h(A^1, A^2), h(A^2, A^1)) \quad (1)$$

where

$$h(A^1, A^2) = \max_{a \in A^1} \min_{b \in A^2} d(a, b) \quad (2)$$

where $dist(a, b)$ is an appropriately defined distance metric between two instances a and b .

However, this metric is quite sensitive to the presence of any outlying point on either bag, whose minimum distance from any of the instances in the other bag may be quite high. Therefore, a modified version of Equation (2) is proposed:

$$h_{(d)}(A^1, A^2) = d_{a \in A^1}^{th} \min_{b \in A^2} dist(a, b) \quad (3)$$

When $d = p$, equations (2) and (3) are equivalent. When $d = 1$, the minimum of individual point distances between the two bags decides the inter-bag distance.

Given a test bag, let χ_R be the number of *references*, i.e., training sample bags that can be considered as neighbours (i.e., having low Hausdorff distance) to the test bag. This can be found by defining a neighbourhood size η_R – this means that the training bags with the lowest η_R distances from the test bag are to be considered as references.

Similarly, let χ_C be the number of *citers*, i.e., training sample bags that would consider the test bag their neighbour. This can be found by defining a neighbourhood size η_C – this means that the training bags for which the test bag falls within the lowest η_C distances from them are to be considered as citers.

Note that the two concepts are not identical – the test bag may be closest to a particular training bag, but from the standpoint of that training bag, there may be other training bags that are closer to it than the test bag.

Now, let χ_R^+ and χ_R^- be the number of positively and negatively labeled bags among the references, and χ_C^+ and χ_C^- be the number of positively and negatively labeled bags among the citers. The predicted label for the test bag B^{new} is +1 if:

$$\frac{\chi_R^+ + \chi_C^+}{\chi_R + \chi_C} \geq \theta, \quad \text{where } \theta = 0.5 \text{ typically} \quad (4)$$

and -1 otherwise. In other words, if there are more positively labeled references and citers for the test bag, its predicted label is positive. See Algorithm 2 for the algorithm pseudocode.

3.1.1. CUSTOMIZING THE CNN MODEL

Optimizing the CNN model typically involves finding the number of references and citers to use (in other words, fix η_R and η_C), as well as the value of the rank d in the Hausdorff distance calculation (empirically, $d = 1$ has been found to be effective in most cases).

However, in our invention we consider the following additional customizations:

1. In problems where we also know the instance labels (but where it is still beneficial to solve the problem as one of multiple instance learning), we could give higher importance to proximity with a positively labeled instance inside a positively labeled bag.

Since the logic behind the CNN algorithm is that the positive examples across bags are likely to fall close to each other in the feature space, these customizations may allow us to exploit such proximity to a greater extent.

2. While comparing the labels of references and citers put together, we could give higher importance to positive bags than negative ones. This may be useful in situations where the cost of misclassification is asymmetric or where the user is primarily interested in optimizing the accuracy on the positive class, while keeping false positives below an acceptable limit.
3. In situations where the feature set describing each instance is quite large, there is the problem of feature selection in order to arrive at a parsimonious model with good generalization ability.

Given the above customizations, it is intuitive that one would need to have a process whereby these parameters are appropriately set for the problem in question. Given that our ultimate objective is to create a classifier that can predict the true label of an unseen test instance bag B^{new} with high accuracy, the ideal combination of parameters ought to be one that maximizes this generalization ability.

In order to estimate generalization ability, we typically use the cross-validation technique. In other words, we take out some of the instance bags in the the training sample (known as the *training subsample*) to use for training, train a model on this part, and test the model on the remaining

Algorithm 2: Citation Nearest Neighbour (CNN) Classifier

Input: *Training sample:* $Train = \{B^1, B^2, \dots, B^T\}$, where each bag B^i contains instances $\{X_1^i, \dots, X_{N_i}^i\}$ and is labeled $Y^i \in \{-1, +1\}$. Each individual instance X_ℓ^i is described by an m -dimensional feature vector $(X_{\ell(1)}^i \dots X_{\ell(m)}^i)$.

Threshold defining references: η_R

Threshold defining citers: η_C

Rank used in Hausdorff distance: d

Feature subset: $S \subseteq \{1 \dots m\}$

Threshold for classification: θ

Unseen instance bag: B^{tst}

Output: *Prediction for* B^{tst} : \hat{Y}^{tst}

begin

function $\hat{Y}^{tst} = CNN (Train, \eta_R, \eta_C, d, S, \theta, B^{tst})$

1. Calculate the $\Lambda_{(T+1) \times T}$ matrix of pairwise distances between instance bags, where:

- **If** $i \leq T$, **then** $\Lambda_{i,j} = H(B^i, B^j)$, **else** $\Lambda_{i,j} = H(B^i, B^{tst})$
- The Hausdorff distance $H(A^1, A^2) = \max(h_{(d)}(A^1, A^2), h_{(d)}(A^2, A^1))$ (see equation 3)
- The distance metric $dist(a, b)$ between instance pairs from two bags is to be calculated on the feature subset S

2. $\chi_R^+ = 0, \chi_C^+ = 0, \chi_R^- = 0, \chi_C^- = 0$

3. **For each** $B^i, i = 1 \dots N$:

- (a) **If** $\Lambda_{T+1,i}$ is within the η_R smallest values in $\Lambda_{T+1, \cdot}$ and $Y^i = +1$, **then** $\chi_R^+ = \chi_R^+ + 1$
- (b) **If** $\Lambda_{T+1,i}$ is within the η_R smallest values in $\Lambda_{T+1, \cdot}$ and $Y^i = -1$, **then** $\chi_R^- = \chi_R^- + 1$
If $\Lambda_{T+1,i}$ is within the η_R smallest values in $\Lambda_{\cdot, i}$ and $Y^i = +1$, **then** $\chi_C^+ = \chi_C^+ + 1$
- (c) **If** $\Lambda_{T+1,i}$ is within the η_R smallest values in $\Lambda_{\cdot, i}$ and $Y^i = -1$, **then** $\chi_C^- = \chi_C^- + 1$

4. Calculate the classifier score for this example as:

$$Score = \frac{\chi_R^+ + \chi_C^+}{\chi_R^+ + \chi_C^+ + \chi_R^- + \chi_C^-}$$

5. **If** $Score \geq \theta$, **then** $\hat{Y}^{tst} = +1$ **else** $\hat{Y}^{tst} = -1$

6. **Return** \hat{Y}^{tst}

end

instance bags (known as the *validation subsample*) as a way of checking its performance on unseen examples. However, in order to avoid sample bias, we need to do this repeatedly and choose different ways to split the training sample into training and validation subsamples.

A systematic way to do this would be to split the dataset into roughly equal sized chunks (say k chunks). In each iteration, one of the chunks is used as the validation subsample while the other chunks together comprise the training subsample. By doing this with each chunk in turn, we ensure that all examples in the training subsample are used in the validation subsample at some point or another. The average performance of the validation subsamples across these iterations is used as a measure of generalization ability (performance on unseen examples). This approach is referred to as k -fold validation in the literature.

An extreme case of this procedure is one where each split contains only one example in the validation subsample – this method of splitting is repeated as many times as the number of examples in the training sample, and the average performance on the validation subsamples across all these splits is reported as an estimate of generalization ability. This approach is called the leave-one-out strategy (see Algorithm 3).

Depending on the size of the dataset (i.e., the number of instance bags), one can choose either the k -fold validation technique or the leave-one-out strategy described above – these are the two strategies recommended in our proposed invention.

The customization process can therefore be described as follows:

1. Consider a set of potential combinations of parameters for the CNN model. These include η_R , η_C , d , relative importance of positive to negative bags while calculating the final label (θ), subset of features to use while calculating the distance metric etc.
2. For each potential combination, estimate the generalization ability using the leave-one-out or similar method.
3. Choose the parameter combination that achieves the best generalization ability.

3.2. Building the ensemble of CNN classifiers

When we consider the method described in Section 3.1.1 above to customize a CNN model, two things become obvious:

- The number of parameters to tune is sufficiently large that the problem of searching through all combinations of parameters may be non-trivial from a computational perspective. Therefore, one may need a smart search algorithm to identify the best combination of parameters from a large possible set.

Algorithm 3: Leave-one-out validation for CNN Classifier

Input: *Training sample:* $Train = \{B^1, B^2, \dots, B^N\}$, where each bag B^i contains instances $\{X_1^i, \dots, X_{N_i}^i\}$ and is labeled $Y^i \in \{-1, +1\}$. Each individual instance X_ℓ^i is described by an m -dimensional feature vector $(X_{\ell(1)}^i \dots X_{\ell(m)}^i)$.

Threshold defining references: η_R

Threshold defining citers: η_C

Rank used in Hausdorff distance: d

Feature subset: $S \subseteq \{1 \dots m\}$

Threshold for classification: θ

Output: *Class +1 (positive) accuracy:* Acc^+

Class -1 (negative) accuracy: Acc^-

Validation outputs: $\hat{Y}^i, i = 1 \dots N$

begin

function $(Acc^+, Acc^-, \hat{Y}^1, \dots, \hat{Y}^N) = LOO(Train, \eta_R, \eta_C, d, S, \theta)$

1. *Initialize* $n^+ = 0, n^- = 0, a^+ = 0, a^- = 0$

2. **For** $i = 1 \dots N$

(a) *Set Training subsample:* $TS^i = Train - B^i$

(b) *Set Validation subsample:* B^i

(c) **If** $Y^i = +1$ **then** $n^+ = n^+ + 1$ **else** $n^- = n^- + 1$

(d) *Call* $\hat{Y}^i = CNN(TS^i, \eta_R, \eta_C, d, S, \theta, B^i)$ (see Algorithm 2)

(e) **If** $Y^i = \hat{Y}^i$ **and** $Y^i = +1$ **then** $a^+ = a^+ + 1$

(f) **If** $Y^i = \hat{Y}^i$ **and** $Y^i = -1$ **then** $a^- = a^- + 1$

3. *Calculate* $Acc^+ = \frac{a^+}{n^+}, Acc^- = \frac{a^-}{n^-}$

4. **Return** $(Acc^+, Acc^-, \hat{Y}^1, \dots, \hat{Y}^N)$

end

- Typically, one wishes to identify a model that maximizes the likelihood of identifying a positively labeled bag correctly, while also minimizing the likelihood of false positives (i.e., negatively labeled bags incorrectly identified). Different parameter combinations are likely to optimize these two metrics in different ways, as they will have different views of the problem space.

There is merit in combining diverse views of the same problem to arrive at a more balanced view overall; therefore, we build an ensemble of CNN classifiers.

We accomplish this by using a multi-objective search heuristic such as NGS-II [Deb et al. \(2002\)](#) to find the optimal CNN parameters. The search algorithm is asked to find the best set of parameters that optimize the following objectives:

1. Maximize the likelihood of classifying a positive instance bag correctly
2. Maximize the likelihood of classifying a negative instance bag correctly

These two objectives are estimated using the leave-one-out method described in [Section 3.1.1](#).

Note that these two objectives may be in conflict in any problem where perfect separability between the classes is not achievable at the given level of solution complexity. Therefore, the multiobjective search algorithm will throw up a set of candidate solutions, each of which optimizes these two objectives at varying degrees of relative importance.

Theoretically, the best possible set is known as a *Pareto frontier* of solutions. Any solution in the Pareto frontier cannot be considered superior to another in the frontier (i.e. if it improves on one objective, it loses on another simultaneously), but can be considered superior to all other solutions available. (Note that in this case, when we use the word solution, we refer to a parameter set for the CNN algorithm, and by performance, we refer to the ability to identify positive and negative bags correctly, as measured using the leave-one-out method.)

In practice, a multi-objective optimizer such as NGS-II will try and arrive at a good approximation to the Pareto frontier, and will output a diverse set of solutions (i.e., parameter combinations for CNN) that optimize the two objectives at varying degrees of relative importance. These solutions constitute the ensemble we wished to construct. The method of combining these solutions is described in [Section 3.2.1](#).

3.2.1. COMBINING THE CNN CLASSIFIERS IN THE ENSEMBLE

As described earlier, we construct an ensemble of CNN models in order to capture diverse views of the problem to be solved. However, the task lies before us to combine these views. The simplest

method of combination would be to let all of these models vote on a test instance bag, and let the majority decide the label. However, it is possible that the optimal method of combination of these diverse views (as represented by the CNN models in the ensemble) calls for a greater degree of complexity than a voting scheme.

Therefore, in our invention, we propose the use of the stacked generalization method [Wolpert \(1992\)](#), wherein we build a second level classifier F , which will combine the predictions of the various CNN models in order to return the final prediction. F can be any two-class classifier such as a support vector machine, random forest etc.

In order to train this classifier, we use the predictions obtained from each member of the ensemble for each instance bag through the validation method described in Section 3.1.1. One can also choose to optimize the parameters of this classifier using the NSGA-II or similar algorithm, as described in Section 3.2 above.

4. Empirical validation

We demonstrate the utility of our proposed method on the *Musk 1* dataset taken from the UCI Machine Learning repository. This dataset describes a set of 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 molecules are judged to be non-musks. The goal is to learn to predict whether new molecules will be musks or non-musks. However, the 166 features that describe these molecules depend upon the exact shape, or conformation, of the molecule. Because bonds can rotate, a single molecule can adopt many different shapes. To generate this data set, the low-energy conformations of the molecules were generated and then filtered to remove highly similar conformations. This left 476 conformations. Then, a feature vector was extracted that describes each conformation.

This many-to-one relationship between feature vectors and molecules lends itself naturally to a multiple instance problem. When learning a classifier for this data, the classifier should classify a molecule as musk if any of its conformations is classified as a musk. A molecule should be classified as non-musk if none of its conformations is classified as a musk [Bache and Lichman \(2013\)](#).

4.1. CNN models

The solution parameters to be optimized are the CNN model parameters, as well as the feature subset used to compute distance between instances. Since this is a large-scale multi-objective optimization problem with objectives where the gradient is ill-defined, we use a direct search method such as a multi-objective genetic algorithm to solve it. Specifically, we use the Non-dominated Sorting

Algorithm 4: Optimizing the parameters for CNN using a multi-objective optimization algorithm

Input: *Training sample:* $Train = \{B^1, B^2, \dots B^N\}$, where each bag B^i contains instances

$\{X_1^i, \dots X_{N_i}^i\}$ and is labeled $Y^i \in \{-1, +1\}$

Output: *CNN classifiers:* $C_{(1)} \dots C_{(J)}$

begin

function $(C_{(1)} \dots C_{(J)}) = MOO(Train)$

Formulation The problem of finding the optimal ensemble of CNN classifiers is stated as follows:

$$\begin{aligned}
 & \max Acc^+(C, Train) \\
 & \max Acc^-(C, Train) \\
 & \text{where} \\
 & C = (\eta_R, \eta_C, d, S, \theta) \quad \text{SeeAlgorithm2} \\
 & (Acc^+, Acc^-, \hat{Y}^1, \dots \hat{Y}^N) = LOO(Train, \eta_R, \eta_C, d, S, \theta) \quad \text{SeeAlgorithm3}
 \end{aligned} \tag{5}$$

Each candidate solution (CNN classifier) is parameterized in terms of the following variables (see Algorithm 2): $(\eta_R, \eta_C, d, S, \theta)$. The goodness of each candidate solution is the leave-one-out validation performance of the classifier, in terms of accuracy in identifying positive and negative bags (see Algorithm 3).

Result A Pareto-optimal set of candidate solutions $C_{(1)} \dots C_{(J)}$, where every pair $C_{(i)}, C_{(j)}$ of solutions is such that, if $Acc_{(i)}^+ > Acc_{(j)}^+$, then $Acc_{(i)}^- < Acc_{(j)}^-$, and vice-versa. This means that, without any additional information that allows us to choose between accuracy on positive bags versus accuracy on negative bags, we cannot choose between any of the solutions in the Pareto-optimal set.

end

Algorithm 5: Building a stacked ensemble of CNN classifiers

Input: *Training sample:* $Train = \{B^1, B^2, \dots, B^N\}$, where each bag B^i contains instances $\{X_1^i, \dots, X_{N_i}^i\}$ and is labeled $Y^i \in \{-1, +1\}$

Output: *CNN classifiers:* $C_{(1)} \dots C_{(J)}$

Final classifier: F

begin

function $(C_{(1)} \dots C_{(J)}, F) = StackEns(Train)$

CNN classifiers Call a multiobjective optimization algorithm (e.g. NSGA-II) to find an optimal ensemble of CNN classifiers (see Algorithm 4):

$$(C_{(1)} \dots C_{(J)}) = MOO(Train)$$

Training sample for stacked ensemble Construct the training sample for generating the second stage classifier, i.e., construct $T2_{N \times J}$ where column $T2_{\cdot j}$ is the set of leave-one-out predictions $(\hat{Y}_{(j)}^1, \dots, \hat{Y}_{(j)}^N)$ obtained for classifier $C_{(j)}$, generated through the following function call (see Algorithm 3):

$$(Acc_{(j)}^+, Acc_{(j)}^-, \hat{Y}_{(j)}^1, \dots, \hat{Y}_{(j)}^N) = LOO(Train, \eta_{R(j)}, \eta_{C(j)}, d_{(j)}, S_{(j)}, \theta_{(j)})$$

Each row $T2_{\cdot i}$ is associated with the class label Y^i for bag B^i .

Final classifier Build a standard 2-class classifier F using the labeled training set $T2$ generated above.

Return $(C_{(1)} \dots C_{(J)}, F)$

end

Genetic Algorithm II (NSGA-II) to optimize the CKNN parameters and feature subset [Deb et al. \(2002\)](#); [Deb \(2001\)](#).

The fitness functions (Class +1 and Class -1 accuracy) are calculated for each solution (i.e., CKNN parameters and feature subset) by considering the average performance on cross-validation samples obtained using the leave-one-out method. This method has been shown to give good estimates of generalization ability [Vapnik \(1998\)](#), and would therefore help us in arriving at the best possible model from a deployment standpoint.

Since NSGA-II is a multi-objective optimization method, its purpose is to generate a *Pareto frontier* of solutions (CKNN models), namely those which represent the best possible trade-off between the various objectives (Class +1 and Class -1 accuracy). Table 1 gives a summary of the results.

Table 1: Citation Nearest Neighbour algorithm results arrived at using the NSGA-II optimization method

Class 0 accuracy	Class 1 accuracy	# Models
100%	91.49%	12
95.56%	95.74%	42
93.33%	97.87%	16
84.44%	100%	30

4.2. Stacked ensemble

We find that the results of the CNN algorithm, tuned as described in Section 4.1 above, do not yet approach the performance level desired by us. We therefore consider using an ensemble approach, whereby we combine the predictions of the various CKNN models arrived at in the final generation of the NSGA-II run. Since these models approximate the Pareto frontier, it is possible that their combination would allow us to come up with a hybrid model that does even better on both objectives. Also, we wish to keep unrestricted, the method of combination of the CKNN predictions; therefore, we use the stacked generalization approach proposed in [Wolpert \(1992\)](#).

We therefore model the second level learning problem as one of mapping the predictions from the last generation of CKNN models to the desired sequence labels. We choose a Support Vector Machine classifier [Cristianini and Shawe-Taylor \(2000\)](#); [Hsu et al. \(2000\)](#) with a Radial Basis Function kernel in order to combine the predictions. In order to optimize the γ and C parameters

of the SVM model, as well as pick the optimal subset of CKNN models whose predictions are to be combined, we again use the NSGA-II algorithm as described in Section 4.1.

4.3. Experimental results

Since NSGA-II generates a Pareto frontier of solutions, a sample of three solutions of the stacked ensemble model is given in Table 2. These results suggest that the stacking layer improves the trade-off between accuracy on the two classes.

Table 2: Selection of solutions arrived at using the stacked ensemble

Class 0 accuracy	Class 1 accuracy	# Models
100%	93.61%	76
97.78%	100%	24

5. Suggestions for further work

The approach described here involves a number of components; therefore, further analysis needs to be done in order to better understand its applicability and correspondence to domain knowledge. From an algorithmic standpoint, one obvious area of further work for the paper is to test it on a diverse set of problems and benchmark it against methods such as those proposed in [Dietterich et al. \(1997\)](#); [Zhou and Zhang \(2007\)](#), as a way of validating the effectiveness of the proposed method.

Furthermore, we have noticed that, for problems with large feature sizes, the computational effort required to arrive at a solution can be quite high. Parallelism, caching and other tactics need to be explored further in order to make this a viable solution.

References

- S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems*, volume 15, 2002.
- B. Babenko. Multiple instance learning: algorithms and applications. Technical report, University of California, San Diego, 2008.
- K. Bache and M. Lichman. UCI machine learning repository, 2013. URL <http://archive.ics.uci.edu/ml>.

- J. Bi and J. Liang. Multiple instance learning of pulmonary embolism detection with geodesic distance along vascular structure. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007.
- J. Bi and J. Liang. Method of multiple instance learning and classification with correlations in object detection. US Patent No. US7822252 B2, 2010.
- L. Breiman and E. Schapire. Random forests. *Machine Learning*, 2001.
- L. Breiman, J. H. Friedman, R. A. Olshem, and C. J. Stone. *Classification and Regression Trees*. CRC Press, London, 1984.
- N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.
- J. Bi D. Wu and K. Boyer. A min-max framework of cascaded classifier with multiple instance learning for computer aided diagnosis. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition*, 2009.
- K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2), 2002.
- Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons., 2001.
- T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:31–71, 1997.
- G. Fung, M. Dundar, B. Krishnapuram, and R. B. Rao. Multiple instance learning for computer aided diagnosis. In *Advances in Neural Information Processing Systems*, 2007.
- C-W. Hsu, C-C. Chang, and C-J. Lin. A practical guide to support vector classification. 2000.
- Balaji Krishnapuram, Vikas C. Raykar, Murat Dundar, and R. Bharat Rao. System and method for multiple-instance learning for computer aided diagnosis. US Patent No. US20090080731 A1, 2009.
- O. Maron and T. Lozano-Perez. A framework for multiple-instance learning. In *Advances in neural information processing systems*, volume 10, 1998.
- R. Bharat Rao, Murat Dundar, Balaji Krishnapuram, and Glenn Fung. System and method for multiple instance learning for computer aided detection. US Patent No. US7986827 B2, 2011.

- V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.
- P. Viola, J. C. Platt, and C. Zhang. Multiple instance boosting for object detection. In *Advances in neural information processing systems*, volume 18, 2006.
- J. Wang and J-D. Zucker. Solving the multiple instance problem - a lazy learning approach. In *17th International Conference on Machine Learning*, 2000.
- D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.
- M.-L. Zhang and Z.-H. Zhou. Multi-instance clustering with applications to multi-instance prediction. *Applied Intelligence*, 31(1):47–68, 2009.
- Z.-H. Zhou and M.-L. Zhang. Solving multi-instance problems with classifier ensemble based on constructive clustering. *Knowledge and Information Systems*, 11(2):155–170, 2007.
- Zhi-Hua Zhou. Multi-instance learning: A survey. Technical report, Nanjing University, 2004.